Quantified Mu-calculus for Control Synthesis

Stéphane Riedweg, Sophie Pinchinat

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Quantified Mu-calculus for Control Synthesis

Stéphane Riedweg — Sophie Pinchinat

N° 4793

Avril 2003

THÈME 1

ISSN 0249-6399 ISRN INRIA/RR--4793--FR+ENG

Quantified Mu-calculus for Control Synthesis

Stéphane Riedweg , Sophie Pinchinat

Thème 1 — Réseaux et systèmes
Projet S4

Rapport de recherche n° 4793 — Avril 2003 — 14 pages

Abstract: We consider an extension of the mu-calculus as a general framework to describe and synthesize controllers. This extension is obtained by quantifying atomic propositions, we call the resulting logic *quantified mu-calculus*. We study its main theoretical properties and show its adequacy to control applications. The proposed framework is expressive : it offers a uniform way to describe as varied parameters as the kind of systems (closed or open), the control objective, the type of interaction between the controller and the system, the optimality criteria (fairness, maximally permissive), etc. To our knowledge, none of the former approaches can capture such a wide range of concepts

Key-words: Mu-calculus, quantified mu-calculus, synthesis, control, supervisory, optimaly criteria

Unité de recherche INRIA Rennes
IRISA, Campus universitaire de Beaulieu, 35042 Rennes Cedex (France)
Téléphone : +33 2 99 84 71 00 — Télécopie : +33 2 99 84 71 71

Mu-calcul quantifié pour la synthèse de controlleurs

Résumé : Nous considérons une extension du mu-calcul pour la description et la synthèse de contrôleurs. Cette extension est obtenue par quantification des propositions atomiques, nous appelons *le mu-calcul quantifié* la logique résultante. Nous étudions ses aspects théoriques et son adéquation aux problèmes de contrôle. Le cadre que nous proposons est expressif: il offre un moyen uniforme de décrire différents paramètres comme le type de système (ouvert ou fermé), l'objectif de contrôle, le type d'interaction entre le contrôleur et le système, des critères d'optimalité (équité, permissivité maximale), etc. A notre connaissance, aucune des approches existantes ne de saisir un si vaste ensemble de concepts.

Mots-clés : Mu-calcul, mu-calcul quantifié, synthèse, contrôleurs, superviseurs, critères d'optimalité

1 Introduction

In the aim of generalizing the theory of control synthesis introduced by Ramadge and Wohnam [1], lot of works use temporal logics as specification [2–4]. All those approaches suffer from substantial limitations: there is no way to impose properties on the interaction between the system and its controller, nor to require optimality of controllers. The motivation of our work is to fill these gaps. We put forward an extension of the mu-calculus well suited to describe general control objectives and to synthesize finite state controllers. The proposed framework is expressive : it offers a uniform way to describe as varied parameters as the kind of systems (closed or open), the control objective, the type of interaction between the controller and the system, the optimality criteria (fairness, maximally permissive), etc. To our knowledge, none of the former approaches can capture such a wide range of concepts.

As in [5–7], we extend a temporal logic (the mu-calculus) by quantifying atomic propositions. We call the resulting logic *quantified mu-calculus*. We study its main theoretical properties and show its adequacy to control applications. We start from alternating tree automata for mu-calculus [8, 9] and we extend their theory using the Simulation Theorem [10, 11, 8] and a projection of automata. The Simulation Theorem states that alternating automata and nondeterministic automata are equivalent. The projection is an adaption of the construction of [12]. The meanings of existential quantifier is defined projecting automata on sets of propositions. Decision procedures for model-checking and satisfaction can therefore be obtained. Both problems are non-elementary when we consider the full logic. We can however display interesting fragments with lower complexity, covering still a wide class of control problems.

The following aims at explaining the applications of our approach to control. We view supervision of systems as pruning the systems' computation trees. Consequently, a controller can be canonically represented by a labeling c of the (uncontrolled) system's computation tree into $\{0, 1\}$, such that the (downwards closed) 1-labeled subtree defines the behavior of the controlled system. For any proposition c, we define a transformation $\|\alpha\|(c)$ of mu-calculus formulas α such that some controller induced restriction of $\mathcal{S}$ satisfies α if and only if $c \wedge \|\alpha\|(c)$ holds of some c-labeling on the computation tree. Labeling enables us to consider the forbidden part of the controlled system. We can in this way derive controllers for large classes of specifications, using a constructive model-checking.

Beyond the capability to specify controllers which only cut controllable transitions, we can more interestingly specify (and synthesize) a *maximally permissive controller* for α; i.e. a controller c such that the c-controlled system satisfies α and no c'-controlled system such that $cut(c) \subsetneq cut(c')$ satisfies α; where $cut(c) \subsetneq cut(c')$ is the mu-calculus formula expressing that the 1-labeled subtree defined by c is a proper subtree of the 1-labeled subtree defined by c'. A maximally permissive controller enforcing α can therefore be specified by the quantified mu-calculus formula:

$$\exists c.\left[\Big(c \wedge \|\alpha\|(c)\Big) \wedge \forall c'.\Big(cut(c) \subsetneq cut(c') \Rightarrow \neg\big(c' \wedge \|\alpha\|(c')\big)\Big)\right]$$

Controllers and maximally permissive controllers for *open systems* [2] may also be specified and synthesized. Such controllers are required moreover to be robust against the environment's policy. Also, the implementation considerations of [13] and decentralized controllers may be formulated in quantified mu-calculus. Not surprisingly, the expressive power of the mu-calculus enables us to deal with fairness.

The rest of the paper is organized as follows : Section 2 presents the logic. Section 3 studies applications to control theory. Algorithms are developed in Section 4, based on the automata-theoretic semantics. Finally, control synthesis is illustrated in Section 5.

2 Quantified mu-calculus

2.1 Syntax

We assume given a finite set of events A, a finite set of propositions AP, and an infinite set of variables $Var = \{X, Y, \dots\}$.

Definition 1. (Syntax of $\mathrm{Q}L_\mu$) *The set of formulas of the quantified mu-calculus on $\Gamma \subseteq AP$, written $\mathrm{Q}L_\mu(\Gamma)$, is defined by the grammar:*

$$\exists \Lambda.\alpha \mid \neg\alpha_1 \mid \alpha_1 \vee \alpha_2 \mid \beta$$

where $\Lambda \subseteq AP$, $\alpha \in \mathrm{Q}L_\mu(\Gamma \cup \Lambda)$, α_1 and α_2 are formulas in $\mathrm{Q}L_\mu(\Gamma)$, and β is a formula of the pure mu-calculus on Γ. We sometimes write $\mathrm{Q}L_\mu$ instead of $\mathrm{Q}L_\mu(\Gamma)$. We recall that the set of formulas of the pure mu-calculus on $\Gamma \subseteq AP$, written $L_\mu(\Gamma)$, is defined by the grammar:

$$\top \mid p \mid X \mid \neg\beta \mid \ <a>\ \beta \mid \beta \vee \beta' \mid \mu X.\beta(X)$$

where $a \in A$, $p \in \Gamma$, $X \in Var$, and β and β' are in $L_\mu(\Gamma)$. Moreover, in order to ensure that fix-points formulas have meanings, it is required that in each formula $\mu X.\alpha(X)$, X occurs under an even number of negation symbols $\neg$ in $\alpha(X)$.
We write $\bot$, $[a]\alpha$, $\alpha \wedge \beta$, $\nu X.\alpha(X)$, and $\forall \Lambda.\alpha$ respectively for negating $\top$, $<a>\ \neg\alpha$, $\neg\alpha \vee \neg\beta$, $\mu X.\neg\alpha(\neg X)$ and $\exists \Lambda.\neg\alpha$. We write also $\xrightarrow{a}$, $\not\xrightarrow{a}$, $\alpha \Rightarrow \beta$, and $\exists x.\alpha$ respectively for $<a>\ \top$, $[a]\ \bot$, $\neg\alpha \vee \beta$, and $\exists\{x\}.\alpha$.

Extending the terminology of mu-calculus, we call *sentences* all quantified mu-calculus formulas without free variables.

Remark 1. Notice that the syntax of $\mathrm{Q}L_\mu$ does not allow quantifiers inside fixed-point terms. Lifting the restriction goes far beyond the scope of this paper and would demand some effort, since in general, fixed-point operators and quantifiers do not commute.

2.2 Semantics

The quantified mu-calculus, as a generalization of the mu-calculus, is also given an interpretation over deterministic transition structures called *processes* in [3] .

Definition 2. *A* process *on $\Gamma \subseteq AP$ is a tuple $\mathcal{S} =< \Gamma, S, s^0, t, L >$, where S is the set of states, $s^0 \in S$ is the initial state, $t : S \times A \to S$ is a partial function called the* transition function *and $L : S \to \mathcal{P}(\Gamma)$ maps states to subset of propositions.*
A process $\mathcal{S}$ is finite *if S is finite and it is* complete *if for all $(a, s) \in A \times S$, $t(s, a)$ is defined.*

Compound processes can be built up by synchronous product.

Definition 3. *The* (synchronous) product *of two processes $\mathcal{S}_1 =< \Gamma_1, S_1, s_1^0, t_1, L_1 >$ and $\mathcal{S}_2 =< \Gamma_2, S_2, s_2^0, t_2, L_2 >$ on disjoint sets Γ_1 and Γ_2 is the process $\mathcal{S}_1 \times \mathcal{S}_2 =< \Gamma, S_1 \times S_2, (s_1^0, s_2^0), t, L >$ on $\Gamma = \Gamma_1 \cup \Gamma_2$ such that:*

- $(s_1', s_2') \in t((s_1, s_2), a)$ *whenever* $s_1' \in t_1(s_1, a)$ *and* $s_2' \in t_2(s_2, a))$,
- $L(s_1, s_2) = L_1(s_1) \cup L_2(s_2)$.

In the sequel, we shall in particular make the product of a process on Γ with another (complete) process on a disjoint set of propositions Λ in order to obtain a similar process on $\Gamma \cup \Lambda$. This is the way in which $\mathrm{Q}L_\mu$ will be applied to solve control problem (see Theorem 1 Section 3).

Definition 4. *A* labeling process *on $\Lambda \subseteq AP$ is simply a complete process $\mathcal{E}$ on Λ. Now, for any process $\mathcal{S} =< \Gamma, S, s^0, t, L >$ with Γ disjoint from Λ, $\mathcal{S} \times \mathcal{E}$ is called a* labeling of $\mathcal{S}$ (by $\mathcal{E}$) on Λ. *We let Lab_Λ denote the set of labeling processes on Λ.*

Notice that labeling $\mathcal{S}$ on $\emptyset$ amounts to unfold process $\mathcal{S}$.

We can now define the semantics of the logic. As expected, the quantification-free fragment of $\mathrm{Q}L_\mu$ coincides with the mu-calculus.

Definition 5. (Semantics of $\mathrm{Q}L_\mu$) *The interpretation of the formulas in $\mathrm{Q}L_\mu(\Gamma)$ is relative to a process $\mathcal{S} =< \Gamma, S, s^0, t, L >$ and a valuation $val : Var \to \mathcal{P}(S)$. This interpretation $[\![\alpha]\!]_{\mathcal{S}}^{[val]} (\subseteq S)$ is defined inductively as follows:*

$$
\begin{array}{l}
[\![\top]\!]_{\mathcal{S}}^{[val]} = S \\
[\![p]\!]_{\mathcal{S}}^{[val]} = \{s \in S \,|\, p \in L(s)\} \\
[\![X]\!]_{\mathcal{S}}^{[val]} = val(X) \\
[\![\neg\alpha]\!]_{\mathcal{S}}^{[val]} = S \setminus [\![\alpha]\!]_{\mathcal{S}}^{[val]} \\
[\![<a> \alpha]\!]_{\mathcal{S}}^{[val]} = \{s \in S \,|\, \exists s' : t(s, a) = s' \, and \, s' \in [\![\alpha]\!]_{\mathcal{S}}^{[val]}\} \\
[\![\alpha \vee \beta]\!]_{\mathcal{S}}^{[val]} = [\![\alpha]\!]_{\mathcal{S}}^{[val]} \cup [\![\beta]\!]_{\mathcal{S}}^{[val]} \\
[\![\mu X.\alpha(X)]\!]_{\mathcal{S}}^{[val]} = \cap\{V \subseteq S \,|\, [\![\alpha]\!]_{\mathcal{S}}^{[val(V/X)]} \subseteq V\} \\
[\![\exists \Lambda.\alpha]\!]_{\mathcal{S}}^{[val]} = \{s \in S \,|\, \exists \mathcal{E} =< \Lambda, E, \varepsilon^0, t', L' > \in Lab_\Lambda, (s, \varepsilon^0) \in [\![\alpha]\!]_{\mathcal{S} \times \mathcal{E}}^{[val \times E]}\}
\end{array}
$$

where the valuation $val \times E : Var \to \mathcal{S} \times \mathcal{E}$ is defined by:
$(val \times E)(X) = val(X) \times E$ for any $X \in Var$.

Notice that, as for the pure mu-calculus fragment, the valuation *val* does not influence the semantics of sentences. Hence, if α is a $\mathrm{Q}L_\mu$-sentence, we shall write $\mathcal{S} \models \alpha$ whenever the initial state of $\mathcal{S}$ belong to $[\![\alpha]\!]_{\mathcal{S}}$.
We show in the end of the section that the set of models of any $\mathrm{Q}L_\mu$ formula is closed under bisimulation, which exhibits the bounds of expressive power of $\mathrm{Q}L_\mu$.

Definition 6. *A bisimulation between two processes* $\mathcal{S}_1 =< \Gamma, S_1, s^0_1, t_1, L_1 >$ *and* $\mathcal{S}_2 =< \Gamma, S_2, s^0_2, t_2, L_2 >$ *is a total binary relation* $R \subseteq S_1 \times S_2$ *such that* $(s^0_1, s^0_2) \in R$, *and for all* $(s_1, s_2) \in R$:

- $L(s^1) = L(s^2)$,
- $\forall a \in A$, $\forall s'_1 \in S_1$, *if* $t_1(s_1, a) = s'_1$, *then there exists* $s'_2 \in S_2$ *such that* $t_2(s_2, a) = s'_2$ *and* $(s'_1, s'_2) \in R$,
- $\forall a \in A$, $\forall s'_2 \in S_2$, *if* $t_2(s_2, a) = s'_2$, *then there exists* $s'_1 \in S_1$ *such that* $t_1(s_1, a) = s'_1$ *and* $(s'_2, s'_1) \in R$.

Proposition 1. *If* $\mathcal{S}_1 =< \Gamma, S_1, s^0_1, t_1, L_1 >$ *and* $\mathcal{S}_2 =< \Gamma, S_2, s^0_2, t_2, L_2 >$ *are bisimilar processes then for any sentence* $\alpha \in \mathrm{Q}L_\mu(\Gamma)$, *we have:*

$$\mathcal{S}_1 \models \alpha \text{ iff } \mathcal{S}_2 \models \alpha$$

Proof. Suppose that $\mathcal{S}_1$ and $\mathcal{S}_2$ are bisimilar processes. We prove the result by induction on α. The basic case, where α is a formula of the pure mu-calculus, is a well know result of the pure mu-calculus. Now, in the case where $\alpha = \exists \Lambda \alpha'$, suppose that $\mathcal{S}_1 \models \alpha$; we will prove that $\mathcal{S}_2 \models \alpha$. By definition of satisfaction, there exists a labeling process $\mathcal{E}$ on Λ such that: $\mathcal{S}_1 \times \mathcal{E} \models \alpha'$. Since bisimulation relations are preserved by product, $\mathcal{S}_1 \times \mathcal{E}$ and $\mathcal{S}_2 \times \mathcal{E}$ are bisimilar. Hence, from the induction hypothesis, $\mathcal{S}_2 \times \mathcal{E} \models \alpha'$ and $\mathcal{S}_2 \models \alpha$. By duality, if $\mathcal{S}_2 \models \alpha$ then $\mathcal{S}_1 \models \alpha$.
The cases $\alpha = \alpha_1 \vee \alpha_2$ and the cases $\alpha = \neg \alpha'$ are straightforward. □

3 Control specifications

This section presents various examples of specifications for control objectives in $\mathrm{Q}L_\mu$. First, a transformation of formulas is defined, which used to link $\mathrm{Q}L_\mu$ model-checking with control problems, as shown by Theorem 1. Variants of the Theorem are then exploited to capture requirements, such as *maximal permissive controllers*, *controllers for open systems*, etc.

Definition 7. *For any sentence* $\alpha \in \mathrm{Q}L_\mu(\Gamma)$ *and for any* $x \in AP$, *the* x-lift of α *is the formula* $\|\alpha\|(x) \in \mathrm{Q}L_\mu(\Gamma \cup \{x\})$, *inductively defined by :*

$$\begin{array}{ll}
\|\top\|(x) = \top & \|p\|(x) = p \\
\|\alpha \vee \beta\|(x) = \|\alpha\|(x) \vee \|\beta\|(x) & \|\neg\alpha\|(x) = \neg\|\alpha\|(x) \\
\|X\|(x) = X & \| <a> \alpha\|(x) = <a> (x \wedge \|\alpha\|(x)) \\
\|\mu X.\alpha\|(x) = \mu X.\|\alpha\|(x) & \|\exists \Lambda \alpha\|(x) = \exists \Lambda \|\alpha\|(x)
\end{array}$$

Definition 8. *Given a process* $\mathcal{S} =< \Gamma, S, s^0, t, L >$ *and some* $x \in \Gamma$*, the* x-pruning of $\mathcal{S}$ *is the process* $\mathcal{S}_{(\rightarrow^x)} =< \Gamma, S, s^0, t', L >$ *such that, for all* $s \in S$ *and* $a \in A$*,* $t'(s,a) = t(s,a)$ *if* $x \in L(t(s,a))$ *or* $t'(s,a)$ *is undefined otherwise.*

Lemma 1. *For any process* $\mathcal{S}$ *on* Γ*, for any* $x \in \Gamma$*, and for any sentence* $\alpha \in \mathrm{Q}L_\mu(\Gamma)$*, we have:* $[\![\alpha]\!]_{\mathcal{S}_{(\rightarrow^x)}} = [\![\|\alpha\|(x)]\!]_{\mathcal{S}}$.

Proof. Straightforward by induction on α. □

Control synthesis as explained in *e.g.* [1, 3, 14, 4] is a generic problem that consists, given a process $\mathcal{S}$ called a plant and a property α not necessary satisfied by this plant, to enforce this property by making the product of the plant with an additional control system $\mathcal{R}$ that restricts its behaviors. $\mathcal{R}$ is called a *controller of* $\mathcal{S}$ *for* α, and the goal is to synthesize such control systems. In this section we focus on the joint specification of control objectives and additional constraints on $\mathcal{R}$ reflecting the way in which this system exerts control. This capability relies on the following theorem.

Theorem 1. *Given a sentence* $\alpha \in \mathrm{Q}L_\mu(\Lambda \cup \Gamma)$*, where* Λ *and* Γ *are disjoint, and a process* $\mathcal{S}$ *on* Γ*, the following assertions are equivalent :*

- *There exists a controller on* Λ *of* $\mathcal{S}$ *for* α.
- $\mathcal{S} \models \exists c \exists \Lambda.\, c \wedge \|\alpha\|(c)$ *where* c *is a fresh proposition.*

Proof. First, suppose that there exists a process $\mathcal{R} =< \Lambda, R, r^0, t, L >$ such that $\mathcal{S} \times \mathcal{R} \models \alpha$. Given $c \in AP \setminus (\Lambda \cup \Gamma)$, we define the labeling process $\mathcal{E} =< \Lambda \cup \{c\}, R \cup \{q_{(\neg c)}\}, r^0, t', L' >$ where $\forall r \in R : L'(r) = L(r) \cup \{c\}$, $L'(q_{(\neg c)}) = \emptyset$, and $\forall a \in A$:

- $t'(q_{(\neg c)}, a) = q_{(\neg c)}$,
- $t'(r,a) = \begin{cases} t(r,a) & \text{if } t(r,a) \text{ is defined} \\ q_{(\neg c)} & \text{otherwise} \end{cases}$

$(\mathcal{S} \times \mathcal{E})_{(\rightarrow^c)}$ or equivalently $\mathcal{S} \times (\mathcal{E})_{(\rightarrow^c)}$ satisfies α, since $\mathcal{R}$ is $(\mathcal{E})_{(\rightarrow^c)}$ without c and c does not occur in α. Using Lemma 1, we conclude that $(\mathcal{S} \times \mathcal{E}) \models c \wedge \|\alpha\|(c)$.
Suppose now that $\mathcal{S} \models \exists c \exists \Lambda . c \wedge \|\alpha\|(c)$. By Definition 5, there is a labeling process $\mathcal{E} \in Lab_{\{c\} \cup \Lambda}$ such that $\mathcal{S} \times \mathcal{E} \models c \wedge \|\alpha\|(c)$. By Lemma 1, $(\mathcal{S} \times \mathcal{E})_{(\rightarrow^c)}$ satisfies α. Then take $\mathcal{R}$ as $(\mathcal{E})_{(\rightarrow^c)}$. □

We illustrate now the use of $\mathrm{Q}L_\mu$ to specify various control requirements. The formula $\exists c . c \wedge \|\alpha\|(c)$ of Theorem 1 is enriched to integrate control rules. In the sequel, we let $<A>= \bigvee_{a \in A} <a>$, $[A] = \bigwedge_{a \in A}[a]$, $Reach_c(\gamma) = \|\mu Y.\, <A> Y \vee \gamma\|(c)$, and $Inv_c(\gamma) = \|\nu Y.[A]Y \wedge \gamma\|(c)$. Also, the x-lift is canonically extended to any conjunction of propositions.

Maximally permissive admissible controller for α. When a system has uncontrollable transitions, denoted by the set of labels $A_{\mathtt{uc}}$, an *admissible controller* for α should not disable any of them. Its existence may be expressed by the formula (1). An admissible controller c for α is *maximally permissive* if no other admissible controller c' for α can cut strictly less transitions than c. Writing $cut(c) \subsetneq cut(c')$ the mu-calculus formula $Inv_c(c') \wedge Reach_{c'}(\neg c)$; this requirement is expressed by the formula (2).

$$\exists c.\, Inv_c([A_{\mathtt{uc}}]c) \wedge c \wedge \|\alpha\|(c) \tag{1}$$

$$\exists c.\, Inv_c([A_{\mathtt{uc}}]c) \wedge c \wedge \|\alpha\|(c) \wedge \Big(\forall c'.\big(Inv_{c'}([A_{\mathtt{uc}}]c') \wedge (cut(c) \subsetneq cut(c'))\big) \;\Rightarrow\; \neg\big(c' \wedge \|\alpha\|(c')\big)\Big) \tag{2}$$

Maximally permissive open controller for α. As studied in [2], open systems take the environment's policy into account : the alphabet A of transitions is a disjoint union of the alphabet $A_{\mathtt{co}}$ of controllable transitions and the alphabet $A_{\mathtt{uc}}$ of uncontrollable transitions, enabled or not by the environment. The *open controller* must ensures α for any possible choice of the environment. This requirement is expressed by the formula (3), where the proposition e represents the environment's policy. The ad-hoc solution of [2] cannot be easily extended to *maximally permissive open controller*. This requirement is expressed by the formula (4).

$$\exists c. Inv_c([A_{\mathtt{uc}}]c) \wedge \forall e.\Big(\big(e \wedge Inv_e([A_{\mathtt{co}}]e)\big) \;\Rightarrow\; \big(c \wedge e \wedge \|\alpha\|(e \wedge c)\big)\Big) \tag{3}$$

$$\begin{aligned}\exists c. Inv_c([A_{\mathtt{uc}}]c) \wedge \Big[&\forall e.\Big(\big(e \wedge Inv_e([A_{\mathtt{co}}]e)\big) \Rightarrow \big(c \wedge e \wedge \|\alpha\|(e \wedge c)\big)\Big)\\ &\wedge \forall c'.\Big(\big(Inv_{c'}([A_{\mathtt{uc}}]c') \wedge (cut(c) \subsetneq cut(c'))\big) \Rightarrow\\ &\exists e'.e' \wedge Inv_e([A_{\mathtt{co}}]e) \wedge \neg(c' \wedge e' \wedge \|\alpha\|(e' \wedge c'))\Big)\Big]\end{aligned} \tag{4}$$

Implementable controller for "non-blocking". Such a controller [13], is an admissible controller which, moreover, selects exactly one controllable transition at a time, and such that, in the resulting supervised system, a final state (given by the proposition P_f) is always reachable. Let $\mathtt{Nblock} = \nu Z.\big((\mu X.P_f \vee <A> X) \wedge [A]Z\big)$ and let $\mathtt{Impl} = \big((\bigvee_{a \in A_{\mathtt{co}}} \stackrel{a}{\rightarrow}) \Rightarrow \bigvee_{a \in A_{\mathtt{co}}}(<a> c \wedge [A_{\mathtt{co}} \backslash \{a\}]\neg c)\big)$; a non-blocking implementable controller of a system $\mathcal{S}$ may be expressed by the formula: $\exists c.\, c \wedge \|\mathtt{Nblock}\|(c) \wedge Inv_c([A_{\mathtt{uc}}]c) \wedge Inv_c(\mathtt{Impl})$

Decentralized controllers for α. The existence of decentralized controllers $\mathcal{R}_1$ and $\mathcal{R}_2$ such that $\mathcal{S} \times \mathcal{R}_1 \times \mathcal{R}_2 \models \alpha$ may be expressed by the formula: $\exists c_1 \exists c_2.(c_1 \wedge c_2) \wedge \|\alpha\|(c_1 \wedge c_2)$

The algorithms for synthesizing controllers rely on automata semantics of $\mathrm{Q}L_\mu$, presented in the next section.

4 Quantified mu-calculus and automata

Automata-theoretic approaches provide the model theory of mu-calculus, and they offer decision algorithms for the satisfiability and the model-checking problems [15, 10, 16–18, 8]. Depending on the approach followed, different automata have been considered, differing mainly in two orthogonal parameters : the more or less restricted kind of transitions, ranging from alternating automata to the subclass of non-deterministic automata, and the acceptance conditions *e.g.* Rabin, Streett, Motkowski/parity.
The class of tree automata for mu-calculus which we shall adapt to $\mathrm{Q}L_\mu$ is the class of *alternating parity automata*, or shortly *simple automata*, considered in [3]. This adaptation is stated by Theorem 2 below which constitutes the main result of this section; the remaining brings back the material needed for its proof.

Theorem 2. (Main result) *For any sentence $\alpha \in \mathrm{Q}L_\mu(\Gamma)$, there exists a simple automaton A_α on Γ such that, for any process $\mathcal{S}$ on Γ:*

$$\mathcal{S} \models \alpha \text{ iff } \mathcal{S} \text{ is accepted by } \mathcal{A}_\alpha$$

Definition 9. (Simple Automata on Processes) *A* simple automaton *on Γ is a tuple $\mathcal{A} =< \Gamma, Q, Q^\exists, Q^\forall, q^0, \delta : Q \times \mathcal{P}(\Gamma) \to \mathcal{P}(Moves(Q)), r >$ where Q is a finite set of states, partitioned into two subsets $Q^\exists$ and $Q^\forall$ of respectively* existential *and* universal *states, $q^0 \in Q$ is the initial state, $r : Q \to I\!N$ is the* parity condition*, and* the transition function δ *assigns to each state q and to each subset of Γ a set of possible* moves *included in $Moves(Q) = ((A \cup \{\epsilon\}) \times Q) \cup (A \times \{\to, \not\to\})$.*

Definition 10. (Nondeterministic Automata on Processes) *A simple automaton is* nondeterministic *if for any set of labels $\Lambda \subseteq \Gamma$, $\delta(q, \Lambda) \subseteq \{\epsilon\} \times Q$ for any $q \in Q^\exists$, and $\delta(q, \Lambda) \subseteq Moves(Q) \setminus \{\epsilon\} \times Q$ for any $q \in Q^\forall$. Moreover, in case when $q \in Q^\forall$, it is required that $(a_1, q_1), (a_2, q_2) \in \delta(q, \Lambda) \cap (A \times Q)$ and $a_1 = a_2$ entail $q_1 = q_2$. Finally, the initial state should be an existential state. A nondeterministic automaton is* bipartite *if for any $\Lambda \subseteq \Gamma$, $\delta(q, \Lambda) \subseteq \{\epsilon\} \times Q^\forall$ for any $q \in Q^\exists$ and $\delta(q, \Lambda) \cap (A \times Q) \subseteq A \times Q^\exists$ for any $q \in Q^\forall$.*

Parity games provide automata semantics. A *parity game* is a graph with an initial vertex v^0, with a partition (V_I, V_{II}) of the vertices, and with a partial mapping r from the vertices to a given finite set of integers. A *play from some vertex v* proceeds as follows: if $v \in V_I$, then player I chooses a successor vertex v', else player II chooses a successor vertex v', and so on ad infinitum unless one player cannot make any move. The *play is winning for player I* if it is finite and ends in a vertex of V_{II}, or if it is infinite and the upper bound of the set of ranks $r(v)$ of vertices v that are encountered infinitely often is even. A *strategy for player I* is a function σ assigning a successor vertex to every sequence of vertices $\vec{v}$, ending in a vertex of V_I. A *strategy σ is memoryless* if $\sigma(\vec{v}) = \sigma(\vec{w})$ whenever the sequences $\vec{v}$ and $\vec{w}$ end in the same vertex. A *strategy for player I is winning* if all play following the strategy from the initial vertex are winning for player I. Winning strategies for player II are defined similarly.

The well-know result of parity games, established in [10, 8], is:

Theorem 3. (Memoryless determinacy) *For any parity game, one of the players has a (memoryless) winning strategy.*

Definition 11. *Given a simple automaton* $\mathcal{A} =< \Gamma, Q, Q^{\exists}, Q^{\forall}, q^0, \delta, r >$ *and a process* $\mathcal{S} =< \Gamma, S, s^0, t, L >$*, we define* the parity game $G(\mathcal{A}, \mathcal{S})$*; where the vertices of player I are in* $Q^{\exists} \times S \cup \{\top\}$ *and the vertices of player II are in* $Q^{\forall} \times S \cup \{\bot\}$*; the initial vertex* v^0 *is* (q^0, s^0)*, the other vertices and transitions are defined inductively as follows. Vertices* $\top$ *and* $\bot$ *have no successor. For any vertex* (q, s) *and for all* $a \in A$*:*

- *there is an* ϵ*-edge to a successor vertex* (q', s) *if* $(\epsilon, q') \in \delta(q, L(s))$*,*
- *there is an* a*-edge to a successor vertex* (q', s') *if* $(a, q') \in \delta(q, L(s))$ *and* $t(s, a) = s'$*,*
- *there is an* a*-edge to a successor vertex* $\top$ *if* $(a, \rightarrow) \in \delta(q, L(s))$ *and* $t(s, a)$ *is defined, or* $(a, \not\rightarrow) \in \delta(q, L(s))$ *and* $t(s, a)$ *is not defined,*
- *there is an* a*-edge to a successor vertex* $\bot$ *if* $(a, \rightarrow) \in \delta(q, L(s))$ *and* $t(s, a)$ *is not defined, or* $(a, \not\rightarrow) \in \delta(q, L(s))$ *and* $t(s, a)$*is defined.*

The automaton $\mathcal{A}$ accepts $\mathcal{S}$ *(noted* $\mathcal{S} \models \mathcal{A}$*) if there is a winning strategy for player I in* $G(\mathcal{A}, \mathcal{S})$.

Like automata on infinite trees [10, 11], simple automata on processes are equivalent to bipartite non-deterministic automata. This fundamental result, due to [8, 3], is called the Simulation Theorem:

Theorem 4. (Simulation Theorem for processes) *Every simple automaton on processes is equivalent to a bipartite nondeterministic automaton.*

A constructive proof of Theorem 2 for $\alpha \in L_\mu$ may be found in [8, 18, 9]. In order to extend this proof to $\mathrm{Q}L_\mu$, we consider projections of automata, which are the semantic counterpart of existential quantification in $\mathrm{Q}L_\mu$. Projections presented here are similar to projections of nondeterministic tree automata presented in [12, 19] : projected automata are obtained by forgetting a subset of propositions in the condition of the transitions.

Definition 12. (Projection) *Let* $\Gamma \subseteq \Gamma'$ *and let* $\mathcal{A} =< \Gamma', Q, Q^{\exists}, Q^{\forall}, q^0, \delta, r >$ *be a bipartite nondeterministic automaton. The* projection of $\mathcal{A}$ on Γ *is the bipartite nondeterministic automaton*
$\mathcal{A}{\downarrow_\Gamma} =< \Gamma, Q^{\exists} \cup Q^{\forall} \times \mathcal{P}(\Lambda), Q^{\exists}, Q^{\forall} \times \mathcal{P}(\Lambda), q^0, \delta{\downarrow_\Gamma}, r{\downarrow_\Gamma} >$
where for all $l \subseteq \Lambda$ *and for all* $l' \subseteq \Gamma$*:*

1. $\forall q \in Q^{\exists} : \delta{\downarrow_\Gamma}\ (q, l') = \{(\epsilon, (q', l)) \,|\, (\epsilon, q') \in \delta(q, l' \cup l)\}$,
2. $\forall q \in Q^{\forall} : \delta{\downarrow_\Gamma}\ ((q, l), l') = \delta(q, l' \cup l)$,
3. $\forall q \in Q^{\exists} : r{\downarrow_\Gamma}\ (q) = r(q)$ *and* $\forall q \in Q^{\forall} : r{\downarrow_\Gamma}\ ((q, l')) = r(q)$.

Theorem 5. (Projection) *Let* $\mathcal{A} =< \Gamma', Q, Q^{\exists}, Q^{\forall}, q^0, \delta, r >$ *be a bipartite nondeterministic automaton. For any process* $\mathcal{S} =< \Gamma, S, s^0, t, L >$ *on* $\Gamma \subseteq \Gamma'$*,* $\mathcal{S} \models \mathcal{A}{\downarrow_\Gamma}$ *if and only if there exists a labeling process* $\mathcal{E}$ *on* $\Lambda = \Gamma' \setminus \Gamma$ *such that* $\mathcal{S} \times \mathcal{E} \models \mathcal{A}$.

Proof. First, suppose $\mathcal{S} \models \mathcal{A}\downarrow_\Gamma$. Let σ be a winning memoryless strategy for player I in the game $G = G(\mathcal{A}\downarrow_\Gamma, \mathcal{S})$ (Theorem 3) and let $V_{II} \subseteq Q^\forall \times \mathcal{P}(\Lambda) \times S$ be the set of nodes from player II in G without $\bot$. Let $\mathcal{E} \in Lab_\Lambda$ be an arbitrary completion of the process $< \Lambda, V_{II}, \sigma(q^0, s^0), t', L' >$, where for any $(q, l, s) \in V_{II}$, $L'(q, l, s) = l$, and for all $a \in A$, $t'((q, l, s), a) = \sigma(s', q')$, for some (unique) (s', q') such that there is an a-arc from $((q, l), s)$ to (q', s') in G. Then, it can be show that we define a winning strategy σ' for player I in the game $G(\mathcal{A}, \mathcal{S} \times \mathcal{E})$ by $\sigma'((s, \sigma(s, q)), q) = \sigma(s, q)$. Reciprocally, suppose $\mathcal{S} \times \mathcal{E} \models \mathcal{A}$ for some $\mathcal{E} \in Lab_\Lambda$. It suffice to show that any memoryless winning strategy for player I in $G(\mathcal{A}, \mathcal{S} \times \mathcal{E})$ defines a memory winning strategy for player I in the game $G(\mathcal{A}\downarrow_\Gamma, \mathcal{S})$. □

Proof of Theorem 2 Let $\alpha \in \mathrm{Q}L_\mu(\Gamma)$ be a sentence, without loss of generality, in prenex normal form $Q_1\Lambda_1 \ldots Q_n\Lambda_n\beta$ where $\beta \in L_\mu$ and the Q_i's form an alternated sequence of existential and universal quantifiers. We proove Theorem 2, by induction on the structure of α. The basic case where $\alpha \in L_\mu$, is dealt with following [8] and [9]. Now, for the case were $\alpha = \exists\Lambda\alpha'$, we let $\mathcal{A}_\alpha$ be the projection on Γ of the bipartite nondeterministic automaton $\mathcal{A}$ that derives from $\mathcal{A}'_\alpha$ by Theorem 4, and we conclude by Theorem 5. For the last case where $\alpha = \forall\Lambda\alpha'$, since parity games are determined (Theorem 3), we can complement, as in [11], any simple automaton. We let hence $\mathcal{A}_\alpha$ be the complement of the automaton $\mathcal{A}_{\exists\Lambda\neg\alpha}$. □

Theorem 2 gives an effective construction of finite controllers on finite processes: given a finite process $\mathcal{S}$ and a sentence $\exists c.\alpha \in \mathrm{Q}L_\mu$ expressing a control problem, we construct the automaton $\mathcal{A}_{(\exists c.\alpha)}$. If we find a memoryless winning strategy in the finite game $G(\mathcal{A}_\alpha, \mathcal{S})$, Theorem 5 gives a finite controller. Otherwise, there is no solution. We can show that the complexity of such problem is $PTIME - complete$ in the size of $\mathcal{S}$ and it is $(k+1)EXPTIME - complete$ in the size of α; where k is the number of alternations of existential and universal quantifiers in α. The result of [2] is retrieved: synthesizing controllers for open systems is $2EXPTIME - complete$ for mu-calculus control objectives.

5 Controller synthesis

This section is devoted to illustrate the constructions on a simple example. The plant $\mathcal{S}$ (to be controlled) is shown in Figure 1. Both states s^0 and s^1 are labeled with the empty set of propositions, thus $\mathcal{S}$ is a process on $\Gamma = \emptyset$. The control objective is the formula $\alpha = \nu Y.\ <b>\ Y \wedge\ <a>\ (\mu X.[a]X)$. Let $\phi = c \wedge \|\alpha\|(c)$, thus $\phi = c \wedge \nu Y.\ <b>\ (c \wedge Y) \wedge\ < a>\ (c \wedge \mu X.[a](c \Rightarrow X))$ and there is a controller of $\mathcal{S}$ for α if and only if $\mathcal{S} \models \exists c.\phi$. The bipartite nondeterministic automaton $\mathcal{A}_\alpha$ is shown in Figure 2, where the following graphical conventions are used: circled states are existential states, while states enclosed in squares are universal states; the transitions between states are represented by edges in $\{a, b, \epsilon\} \times \mathcal{P}(\{c\})$; the other transitions are represented by labeled edges from states to the special boxe containing the symbol $\rightarrow$. The rank function maps q^0 to 2 and q_2 to 1. The projected automaton $\mathcal{A}_\phi\downarrow_\emptyset$ is shown in Figure 3, using similar conventions. Note that all transitions are labeled in $\{a, b, \epsilon\} \times \{\emptyset\}$, since $\mathcal{A}_\phi\downarrow_\emptyset$ is an automaton on $\Gamma = \emptyset$, but all

universal states are now labeled in $Q \times \mathcal{P}(\{c\})$, as a result of the projection. Now, $\mathcal{S} \models \exists c.\alpha$ if and only if $\mathcal{A}_\alpha \downarrow_{\{c\}}$ accept $\mathcal{S}$ and this condition is equivalent to the existence of a winning strategy for player I in the finite parity game $G(\mathcal{A}_\alpha \downarrow_{\{c\}}, \mathcal{S})$ of Figure 4. Clearly, player I has an unique memoryless wining strategy σ, that maps the vertex (q_2, s^0) to $(q'_2, \emptyset, s^0)$. The labeling process $\mathcal{E}$ on $\{c\}$ derived from σ is shown in Figure 5. Four states and transitions between them are first computed, yielding an incomplete process on $\{c\}$. A last state c is then added so as to obtain a complete process. The dashed transitions (and all dead transitions) are finally suppressed to yield the synthesized controller.

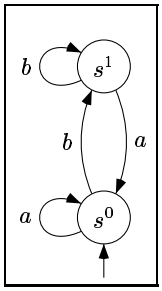

Fig. 1. The plant $\mathcal{S}$

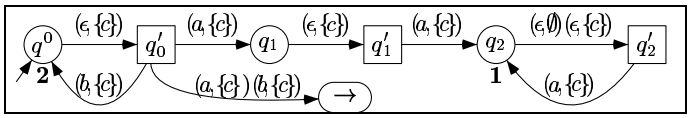

Fig. 2. The nondeterministic automaton $\mathcal{A}_\phi$

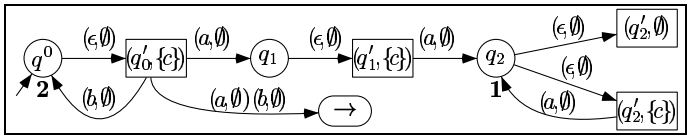

Fig. 3. The nondeterministic automaton $\mathcal{A}_{\exists c.\phi}$

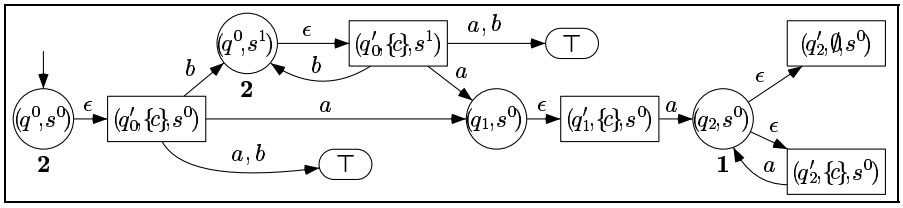

Fig. 4. The game $\mathcal{G}(\mathcal{A}_{\exists c.\phi}, \mathcal{S})$

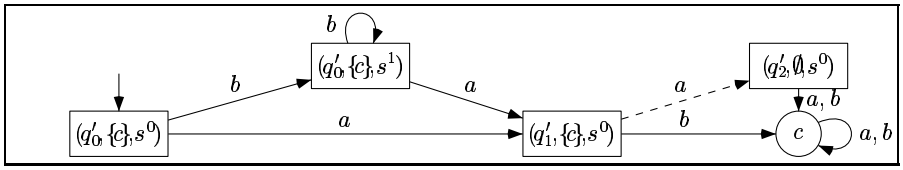

Fig. 5. The labeling process and the controller

6 Conclusion

The logical formalism we have developed allows to synthesize controllers for a large class of control objectives. All the constraints, as maximally permissive controllers or admissible ones for open systems, are formulated as objectives. As it is, the class of controllers is left free and we cannot, for example, deal with partial observation. The recent work of [3] offers two constructions that we can use to interpret the quantified mu-calculus relatively to some fixed classes of labeling processes. The first construction, the quotient of automata, forces the labeling processes to be in some mu-calculus (definable) class. It can be seen as a generalization of the automata projection, and used instead. The quantified mu-calculus could hence be extended by constraining each quantifier to range over some mu-calculus class. Nevertheless, the class of controllers under partial observation being undefinable in the mu-calculus, we need to consider the second construction of [3]: the quotient of automata over a process exhibits a controller under partial observation inside some mu-calculus class (when it exists). The outermost quantification of a sentence is then made relative to some class of partial observation. Therefore, we can seek a controller under partial observation for open systems, but we cannot synthesize a maximally permissive controller among the controllers under partial observation.

References

1. Ramadge, P.J., Wonham, W.M.: The control of discrete event systems. Proceedings of the IEEE; Special issue on Dynamics of Discrete Event Systems **77** (1989) 81–98
2. Kupferman, O., Madhusudan, P., Thiagarajan, P., Vardi, M.: Open systems in reactive environments: Control and synthesis. In: Proc. 11th Int. Conf. on Concurrency Theory. Volume 1877 of Lecture Notes in Computer Science., Springer-Verlag (2000) 92–107
3. Arnold, A., Vincent, A., Walukiewicz, I.: Games for synthesis of controllers with partial observation. to appear in Theoretical Computer Science (2003)
4. Vincent, A.: Synthèse de contrôleurs et stratégies gagnantes dans les jeux de parité. In Hermès, ed.: Modélisation des systèmes réactifs. (2001) 87–98
5. Sistla, A., Vardi, M., Wolper, P.: The complementation problem for Buchï automata with applications to temporal logic. Theoretical Computer Science **49** (1987) 217–237
6. Kupferman, O.: Augmenting branching temporal logics with existential quantification over atomic propositions. JLC: Journal of Logic and Computation **9** (1999)
7. Patthak, A.C., Bhattacharya, I., Dasgupta, A., Dasgupta, P., Chakrabart, P.P.: Quantified computation tree logic. Information Processing Letters **82** (2002) 123–129
8. Arnold, A., Niwinski, D.: Rudiments of mu-calculus. North-Holland (2001)
9. Walukiewicz, I.: Automata and logic. In: Notes from EEF Summer School'01. (2001)
10. Emerson, E.A., Jutla, C.S.: Tree automata, mu-calculus and determinacy. In: Proceedings 32nd Annual IEEE Symp. on Foundations of Computer Science, FOCS'91, San Jose, Puerto Rico, 1–4 Oct 1991. IEEE Computer Society Press, Los Alamitos, CA (1991) 368–377
11. Muller, D.E., Schupp, P.E.: Simulating alternating tree automata by nondeterministic automata: New results and new proofs of the theorems of Rabin, McNaughton and Safra. Theoretical Computer Science **141** (1995) 69–107

12. Rabin, M.O.: Decidability of second-order theories and automata on infinite trees. Trans. Amer. Math. Soc. **141** (1969) 1–35
13. Dietrich, P., Malik, R., Wonham, W., Brandin, B.: Implementation considerations in supervisory control. In Caillaud, B., Darondeau, P., Lavagno, L., Xie, X., eds.: Synthesis and Control of Discrete Event Systems. Kluwer Academic Publishers (2002) 185–201
14. Bergeron, A.: A unified approach to control problems in discrete event processes. Theoretical Informatics and Applications **27** (1993) 555–573
15. Emerson, E.A., Sistla, A.P.: Deciding full branching time logic. Information and Control **61** (1984) 175–201
16. Emerson, E.A., Jutla, C.S., Sistla, A.P.: On model-checking for fragments of mu-calculus. In: Proc. 5th International Computer Aided Verification Conference. (1993) 385–396
17. Streett, R.S., Emerson, E.A.: The propositional mu-calculus is elementary. In: Proc. 11th ICALP, Antwerpen, LNCS 172, Springer-Verlag (1984) 465–472
18. Kupferman, O., Vardi, M.Y., Wolper, P.: An automata-theoretic approach to branching-time model checking. Journal of the ACM **47** (2000) 312–360
19. Thomas, W.: Automata on infinite objects. In Leeuwen, J.v., ed.: Handbook of Theoretical Computer Science, vol. B. Elsevier Science Publishers (1990) 133–191

Unité de recherche INRIA Rennes
IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)

Unité de recherche INRIA Futurs : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)
Unité de recherche INRIA Lorraine : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)
Unité de recherche INRIA Rhône-Alpes : 655, avenue de l'Europe - 38330 Montbonnot-St-Martin (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)
Unité de recherche INRIA Sophia Antipolis : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399